

194-813-SI4-003

EOSDIS Core System Project

DADS Prototype One FSMS Product Operational Evaluation

May 1994

Hughes Applied Information Systems
Landover, Maryland

DADS Prototype One FSMS Product Operational Evaluation for the ECS Project

May 1994

Prepared Under Contract NAS5-60000

APPROVED BY

Stephen Fox /s/	5/24/94
Steve Fox, SDPS Office Manager	Date
EOSDIS Core System Project	

Hughes Applied Information Systems
Landover, Maryland

This page intentionally left blank.

Preface

This document contains the prototype results report for DADS Prototype One. This document is submitted as required by the ECS Statement of Work, Section 3.3.3.3, and does not require Government approval. The results of this prototype will also be documented in the Prototyping and Studies Final Report (DID 333/DV3).

For additional technical information pertaining to the DADS prototype, contact Tom Smith, SDPS-DADS system engineer, at (301) 925-0467 or on email as tsmith@eos.hitc.com.

Questions concerning the control or distribution of this document should be addressed to:

Data Management Office
The ECS Project Office
Hughes Applied Information Systems, Inc.
1616A McCormick Dr.
Landover, MD 20785

This page intentionally left blank.

Contents

Preface

1. Introduction

1.1	Identification	1-1
1.2	Summary	1-1
1.3	Purpose	1-1
1.4	Parent Document	1-2
1.5	Applicable Documents	1-2

2. Hardware Configuration

3. FileServ 2.1.6 Evaluation

3.1	Software Configuration.....	3-1
3.2	Data Distribution.....	3-2
3.3	Operational Evaluation of Features.....	3-3
3.3.1	General	3-3
3.3.2	Startup/Shutdown	3-3
3.3.3	Gui Capabilities.....	3-4
3.3.4	Command Line Functions.....	3-4
3.3.5	Placing Files under FileServ Control	3-4
3.3.6	File Migration and Staging	3-5
3.3.7	Disk Cache Management	3-5
3.3.8	Ease of File Ingest.....	3-6
3.3.9	Concurrent File Write.....	3-6
3.3.10	Ease of File Retrieval - Sequential, Random, Partial	3-7
3.3.11	Concurrent File Read.....	3-7
3.3.12	File Deletion and Tape Repacking	3-7
3.3.13	Accidental File Deletion - Trashcan.....	3-8
3.3.14	Media Operations	3-8
3.3.15	System Logging Functions	3-8

3.3.16	System Reports	3-9
3.3.17	Degraded Mode Operations.....	3-9
3.3.18	Applications Program Interface (API).....	3-9
3.3.19	Identified Problems.....	3-9

4. Convex Unitree 1.75.15 Evaluation

4.1	Software Configuration.....	4-1
4.2	Convex UniTree: Enhancements.....	4-2
4.2.1	Regular File Disk Cache.....	4-2
4.2.2	Rcp Support.....	4-2
4.2.3	Volume Mount-Ahead	4-2
4.2.4	3480 Fast Seek	4-2
4.2.5	Logical Volume Dismounts	4-3
4.2.6	Parallel Copy Writes	4-3
4.2.7	Parallel tapemovrs.....	4-3
4.2.8	Namesrvr Performance	4-3
4.2.9	Repacker Disk Cache and Tape Drive Throttling	4-3
4.2.10	FTP Mstage.....	4-3
4.2.11	UniTree Path Sockets Parameter.....	4-3
4.2.12	Additional Parameters in UniTree.Conf.....	4-4
4.3	Convex Known Problems.....	4-4
4.3.1	More than 6 Ultranet Connections.....	4-4
4.3.2	Automatic Tape Repacker	4-4
4.3.3	Disksrvr Startup Panic Message.....	4-4
4.3.4	Mass Staging (mstage) Problem.....	4-4
4.3.5	Full Disk Cache Problem.....	4-4
4.4	Data Distribution.....	4-5
4.5	Operational Evaluation of Features.....	4-5
4.5.1	General	4-5
4.5.2	Startup/Shutdown	4-6
4.5.3	Placing Files under UniTree Control.....	4-6
4.5.4	File Migration and Staging	4-6
4.5.5	Disk Cache Management	4-7

4.5.6	Ease of File Ingest.....	4-7
4.5.7	Concurrent File Write.....	4-7
4.5.8	Ease of File Retrieval - Sequential, Random, Partial.....	4-8
4.5.9	Concurrent File Read	4-8
4.5.10	File Deletion - Trashcan.....	4-8
4.5.11	Accidental File Deletion - Trashcan.....	4-8
4.5.12	Media Operations	4-8
4.5.13	System Logging Functions	4-9
4.5.14	System Reports	4-9
4.5.15	Degraded Mode Operations.....	4-9
4.5.16	Applications Program Interface (API).....	4-9

5. Conclusions

6. Ancillary Data

6.3	FileServ Error Log	6-4
6.5	UniTree Error Log	6-8
6.5.1	Problem 1 - Spu Disk Corruption.....	6-8
6.5.2	Problem 2 - Drive Parameter Mismatch.....	6-9
6.5.3	Problem 3 - Inaccurate Transfer Rates.....	6-9

Tables

3-1.	FileServ Evalulation - Data Distribution.....	3-2
4-1.	UniTree Evaluation Data Distribution.....	4-5
6-1.	FileServ Command Summary	6-1
6-2.	FileServ Queries and Reports.....	6-3
6-3.	FTP & UniTree FTP Commands.....	6-6
6-4	Convex UniTree Unique Commands	6-6

Abbreviations And Acronyms

This page intentionally left blank.

Introduction

1.1 Identification

This DADS Prototype One Results Report is prepared for the Earth Observing System Data and Information Systems (EOSDIS) Core System (ECS) project, contract number NAS5-60000.

1.2 Summary

Based on the results of Prototype 1, "Unix resident" FSMS products appear to offer a number of advantages over "separate file system based" FSMS products. The FileServ product offered significant advantages over UniTree in the areas of:

- System Administration - Monitoring Archive Activity and Software Parameter Modifications
- Disk Cache Management - Cache Full or Almost Full Procedures
- File Ingestion & Retrieval - Data I/O via the Archive
- File Control - Specifically in the Areas of File Migration and Disk Cache Deletion
- File Tracking - Creation, Physical File Location and File Descriptor Data
- Degraded Mode Operations - Device Failure or Routine Preventive Maintenance
- Error Tracking - Central Error Logging

The advantages listed above represent observed differences in an Evaluation and Test Environment. Product testing in an Operational Environment is required to further validate these observations. In addition, the capabilities of specific implementations of UniTree will vary between OEM vendors. Other implementations of UniTree may perform better or worse than Convex UniTree in one or more areas.

1.3 Purpose

The primary purpose of Prototype 1 was to functionally evaluate the capabilities of the two basic types of File Storage Management Systems (FSMS) on the market today. These types are: (1) FSMS products that reside within the Unix File System (UFS) and (2) products that reside in a separate file system within or in conjunction with UFS.

Native UFS products provide users with all of the file modification and access capabilities inherent in UFS and transparent movement of files between primary (disk) storage to secondary (tape) storage. File movement is controlled by system parameters and/or by user requests.

Separate file system products require users to move files from working directories to a product disk cache. Files are then automatically moved between primary storage to secondary storage. File movement is controlled by system parameters and/or by user requests.

This prototype evaluation was conducted using a single hardware and operating system configuration. The focus of this effort was not to create a performance benchmark, but instead to emphasize the functional capabilities and how these capabilities mapped to ECS Program Requirements.

1.4 Parent Document

February 1993 ECS Statement of Work

1.5 Applicable Documents

The following documents are applicable to this document:

193-707-PP1-002	ECS Prototype Results Review, submitted December 1993
193-216-SE1-001	ECS Requirements Specification, submitted February 1994
193-317-DV1-001	ECS Prototyping and Studies Plan, submitted May 1993
193-318-DV3-005	ECS Prototype and Studies Progress Report, submitted November 1993

2. Hardware Configuration

The following hardware configuration was used to test a representative FSMS from each market type.

HOST:	Convex C3220 Dual CPU System
DISK SPACE:	20 GB
OS VERSION:	10.1.2 initially, upgraded to 10.1.4 - 11/10/93
MAIN MEMORY:	2 GB

ROBOTIC UNIT:	STK WolfCreek 9360 Silo
CARTRIDGE FORMAT:	3480
CARTRIDGE CAPACITY:	800 Cartridges
VOLUME CAPACITY:	200 MB/Cartridge
TAPE TRANSPORTS:	4
DATA TRANSFER INTERFACE:	Block Mux

ACSLs HOST:	Sun SparcStation 330
ACSSA SOFTWARE:	
Release: 4.00	Variant: sun2.0

This page intentionally left blank.

3. FileServ 2.1.6 Evaluation

3.1 Software Configuration

The first software product evaluated was FileServ 2.1.6 from the E-Systems Modular Automated Storage Systems (EMASS) Group. The FileServ product maintains files within the Unix File System. Initial industry studies indicated that FileServ was one of the more mature and robust products in the native UFS category. EMASS provided a Technical Assistance Specialist to configure the Convex File System and install and test the FileServ product. EMASS also provided training materials and 3 days of Software Training.

Users are well insulated from the internals of the FileServ product. The FileServ Administrator has control over the majority of the parameters in the storage system. Modifications to the system parameters require editing of associated configuration files. Direct product access is accomplished via a configurable X-Windows/Motif Graphical User Interface (GUI) and/or a command line interface. User access is accomplished via standard Unix File Commands (e.g. cat <Filename>). FileServ utilizes an Ingres Database to record the File Name, associated header data, and the file location. The administrator establishes relationships between specific UFS directories and the FileServ product. This relationship is referenced using a DataClass name. Once a relationship exists, files in the specified directory will be managed by FileServ transparently.

The following FileServ features were evaluated:

- Startup/Shutdown
- Placing Files Under FileServ Control
- GUI Capabilities
- Command Line Functions
- Ease of File Ingest
- Concurrent File Write
- Ease of File Retrieval
- Concurrent File Read
- Media Operations
- Storage Device Management
- System Logging Functions
- System Reports
- Fault Detection & Isolation
- Partial File Retrieval

- Accidental File Deletion
- Degraded Mode Operations

3.2 Data Distribution

Table 3-1. FileServ Evaluation - Data Distribution

Class Name	# of Files	File Size(s)	# of Tapes Used
A	40	100 MB	20
B	400	10 MB	20
C	4000	1 MB	25
D	6000	100 KB	25
E	99	100 MB, 1 MB	9
F	908	100 MB, 10 MB	11
G	9210	100 MB, 100 KB	14
H	1040	10 MB	13
I	30	1 KB, 200MB, 300MB, 500 MB, 1GB	34
J	5000	1 KB	8
K	5000	1 KB	4
L	5000	1 KB	5
M	5000	1 KB	4
N	5000	1 KB	4
O	5000	1 KB	5
P	5000	1 KB	4
Q	5000	1 KB	5
R	5000	1 KB	5
T	4500	1 KB	5
Misc.	132	Other Test Files	15
TOTAL	71359	71680 Possible Inodes	235/600

This table illustrates the number, size, and distribution of files used for FileServ 2.1.6 testing. The Class Name corresponds to both a specific UFS directory and a FileServ DataClass. A DataClass is a file clustering mechanism implemented by FileServ. Data files will only reside on tapes associated with their DataClass ID. Data from different clusters is placed on different tapes.

A total of 71,359 files were placed under FileServ control. This number was limited by the available inodes on the FileServ data partition. The total inodes in the data partition was 71,680. This was the default number of inodes allocated by Convex OS for a partition of the specified size. Specific OS parameters could have been modified to increase the number of inodes allocated in the data partition. These measures were not deemed necessary for this evaluation.

FileServ allows the FileServ Administrator to decide what amount of the available system resources FileServ will use. The configuration tested placed 3/4 of the WolfCreek's capacity (600 - 3480 tapes) under FileServ Control. FileServ was allowed to use all four (4) tape drives as they were needed.

3.3 Operational Evaluation of Features

3.3.1 General

All users found the FileServ product very easy to use. Even users who were not present for the entire software training session were able to begin using FileServ with minimal assistance from other team members. For the purpose of this section, the term "user" is used generically to indicate members of the prototyping team and will be limited to command line interaction with FileServ unless otherwise indicated..

FileServ software allows the FileServ Administrator to restrict the commands available to a command-line user. These restrictions would normally take on a three-tier control structure. Users have the most limited set of capabilities. Operators have all "user commands" and some additional commands. The FileServ Administrator has all available commands. To reduce dependencies during this evaluation, all prototype team members either had the privileges of a FileServ Administrator or the capability to login as a FileServ Administrator.

The product documentation is clear and concise. Each specific function in the GUI is well illustrated and the command line features and argument options are well described. The only comment in this area is that the documentation lacks specific examples of commands and what the resulting output would be.

Specific tuning and operations optimization is possible using the numerous parameters provided by FileServ. This evaluation was performed using default parameters with a few exceptions noted later. Optimizations based on a specific user profile and/or on hardware capabilities & characteristics were avoided. Tuning was avoided except in the case of operational problems. Any such problems will be detailed below.

FileServ supports both Automated Tape Libraries (ATLs) and a manual archive. The manual archive management and operations capabilities were not explicitly tested.

The default number of files that can be stored on a 3480 cartridge is two hundred (200). This number was modified during the evaluation to one thousand (1,000) thus allowing more files on a single volume. This permitted the prototype team to evaluate streaming tape performance for large numbers of small files. At this point, it is unclear what file sizes will be stored by the FSMS products in ECS. Consequently, multiple file sizes and distributions were examined. Changing this default value resulted in several problems which are discussed separately.

3.3.2 Startup/Shutdown

FileServ relies on the Ingres Database product for much of its data mapping between filenames & inodes, DataClasses, and tape volumes. Any user with the appropriate privilege can startup and shutdown the FileServ system. Ingres requires Ingres account privileges to startup and shutdown

the database. The normal startup sequence is Ingres followed by FileServ. During a normal shutdown this sequence is reversed.

minutes. The FileServ product was operational in 20 seconds on average. A FileServ shutdown took less than 1:00 minute, and an Ingres sUsage: On the average: Ingres started up and was running properly within 2:00 shutdown took less than 2:00 minutes. Times mentioned after this point, unless otherwise indicated, are based on a combination of FileServ & Ingres, since these products function together to form the FileServ FSMS.

3.3.3 Gui Capabilities

FileServ is supplied with a configurable X-Windows/Motif GUI interface. All functions available on the command line can also be performed from the GUI. The GUI is configured by editing a button file. The FileServ Administrator can comment out certain buttons, thus removing those functions from a user's GUI. The button file is then provided to the user and appropriate paths are set to locate the GUI interface button file and executables.

Usage: The GUI was used extensively to create directory and DataClass relations. One annoying feature of this interface was a "Command Successfully Completed" window, which came up after each command was completed. The user was required to "click " OK on this window before he could proceed with the remainder of the command. This feature has been deleted from FileServ 2.2.x.

3.3.4 Command Line Functions

FileServ provides a robust set of command line functions. While all command line functions are available from the GUI, in some instances it was easier to use the command line for some functions. (A command summary is included in Appendix A.)

Usage: The command line interface was used most during this evaluation. The associated parameter syntax. was initially difficult, however, frequent use of the commands eliminated this problem. In addition, manual pages were on-line and available for each command.

3.3.5 Placing Files under FileServ Control

To place files under FileServ control, the user first establishes a relationship between one or more UFS directories and the FileServ software. During this process a DataClass Relationship is normally established and a limit to the number of tapes available to the DataClass is also established. Several other parameters concerning media type, whether a file can span multiple tapes, minimum time a tape will remain on disk, etc., are also established at this time. These parameters can be user tailored or established by default. The majority of these parameters can be changed at any time by the user. Once the appropriate relationships have been established, any files placed in these directories are automatically under FileServ control.

3.3.6 File Migration and Staging

The user may explicitly store or retrieve a file by issuing the *fsstore* <filename> and *fsretrieve* <filename> commands respectively. Migration will also occur based on policies established by the FileServ Administrator. Within FileServ, a policy is an instruction which will operate all files that meet the specific criteria established in the policy parameter list. Policies can be established at the DataClass level and they can be run either by the administrator explicitly or they can be scheduled to run via "*cron*". It is unnecessary for a user to know specific commands to utilize files under FileServ control. If a file is not resident on disk, any file operation (e.g. *vi* <filename>) will result in a staging operation from tape.

Usage: Initially, some *crontab* storage policies were used. Unfortunately, these policies did not allow full control over the order of files stored on tape. All candidate files¹ were copied to tape via some internal FileServ scheduling mechanism. This mechanism did not take into account sequential file naming or file creation date. Rather, the storage appeared to be based roughly on file size. The use of wildcards (e.g. *fsstore* *, on a directory) during storage operations lead to a tape swapping problem discussed in "Concurrent File Write" below. In both cases, the FileServ logging functions did not differentiate the storage times for individual files. For this reason, explicit storage and retrieval commands were used during most of this evaluation.

Problem: Found a minor inconvenience in Convex OS with regard to the use of wildcards. Convex OS will return "argument too long" if a wildcard operation (e.g. *fsstore* *) is attempted on a directory containing a "large" number of files. The actual number of files required was never determined, but tests indicate the number is greater than one hundred (100). The problem seemed to occur primarily when the directory name space was not sequential. (e.g. A, B, C, J, K, L, M,...). This problem was avoided by decreasing the number of files searched by a wildcard command. (e.g. our file format was: FS123456.DAT. Specifying FS1234*.DAT allowed wildcard operations on a smaller file subset of the directory.)

3.3.7 Disk Cache Management

FileServ manages the disk cache via a set of disk utilization "*watermarks*" which are selected by the FileServ Administrator. These parameters are specified in the *fstab* file of ConvexOS when the associated disk partition is created. An operating system callout is generated when each of these *watermarks* is reached. FileServ monitors these callouts and takes appropriate action if the callout references a partition being used by FileServ. The **high block** is determined by the FileServ Administrator based on the user profile. The high block represents the highest amount of disk space that can be used prior to automatic space reclamation activities by the FSMS. (e.g. The FileServ Administrator decides 75% will be used for the high block in a one hundred Gigabyte (100 GB) disk cache.. Thus, the high block is reached when 75 GB of the disk cache are in use.) When the high block is reached, FileServ creates a list of candidate files that have been migrated to tape but not truncated from disk². The **low block** indicates that the available disk space is low. (e.g. The FileServ Administrator decides 90% will be used for the low block. When the disk

¹Files that have exceeded their minimum disk residency time.

²The term truncate is used because the file/inode association remains on disk. Only the data blocks are removed or truncated from the file.

cache reaches 90 GB in use, space in the disk cache is low.) When the low block is reached, an automatic truncation policy is run against the candidate list created when the high block was reached. This policy continues to run until disk utilization is reduced to the high block limit. If the limit is not reached, an automatic storage policy is run against all files that have met their criteria for migration. Then a truncation policy is run against these recently migrated files. Should the **no block** watermark be reached, then no space is available on disk. (e.g. No block is 100% or the disk cache is completely full.) The policies performed when a low block was reached are performed again. If the high block limit is not reached by these policies, a truncation policy ignoring the minimum time a file remains on disk is initiated until the high block is reached.

Usage: As previously mentioned, explicit file operations were performed on the disk files. The watermark features of disk cache management were not tested. In part, this was to simulate the anticipated ECS operational environment. Files will be immediately migrated to tape for backup purposes and they may then reside for a period of time on disk, or they may be immediately truncated.

3.3.8 Ease of File Ingest

File ingestion into the FileServ system was very easy.

Usage: A test generator was developed for use with FSMS products. The test data was generated in directories already under FileServ control. Policy storage functions, and wildcard storage operations were tested. Explicit storage and truncation operations were used for the most part.

3.3.9 Concurrent File Write

The FileServ Administrator can allocate any number of the available tape drives in the system to FileServ. FileServ uses these devices as needed. FileServ also balanced its resource utilization by using each available drive in succession. In this manner, no specific drive was used more than any other.

Usage: During wildcard storage operations all available tape drives were used by FileServ. Multiple explicit storage operations by separate users were also tested. Each user storage operation occurred independently of any other.

FSMS access time was relatively constant for write operations. On the average, FSMS access/update time was eight (8) seconds for a 100 MB file. Variance was no more than one (1) second for smaller or larger file writes.

Problem: The only problem identified during this portion of testing was that FileServ 2.1.6 assumes each storage request is independent of any other. Thus it would mount, store, then dismount each tape after one file operation was completed. If multiple storage operations occurred, two tapes were used by FileServ in the following sequence until the tapes reached their capacity: Mount Tape 1 - goto EOT - Write File - Rewind - Dismount Tape 1 - Mount Tape 2 - goto EOT - Write File -

Rewind - Dismount Tape 2 - Mount Tape 1...etc., until all files were copied. This problem will be corrected in Version 2.2 with a delayed dismount feature³.

3.3.10 Ease of File Retrieval - Sequential, Random, Partial

Usage: The integrity of each data file was checked after each retrieval operation. No data errors were ever found. It is possible to retrieve the entire contents of a tape though the syntax of this command is not really described in the FileServ documentation. For the most part, explicit whole file random retrievals from different volumes in the same DataClass, and random retrievals of files on the same volume were tested. If the delayed dismount feature scheduled for Version 2.2 had been available in this release, there would probably have been a measurable difference in random retrievals from multiple volumes as opposed to the same volume. FileServ also offers a partial file retrieval capability. This function allows a user to retrieve any portion of contiguous bytes from a specified file. The partial file retrieval parameters require the user to specify a new file name for the retrieved file segment. This feature was tested and appears to work as advertised.

FileServ logs indicate the FSMS uses three (3) seconds to process a whole file retrieval, on the average, and an additional two (2) seconds of processing as the retrieval completed. EMASS technical support stated that the additional time is probably a combination of a database update and an inode update to indicate the file is now on disk. Partial file retrievals required the user to specify a new name for the subset of a data file requested. The same three (3) seconds of FSMS processing was required for a partial retrieval but the additional two (2) seconds did not appear in the case of partial retrievals. Since this file did not previously exist, it was not necessary to update file residency information.

3.3.11 Concurrent File Read

FileServ will allow concurrent read operations to occur on all devices available to the FSMS.

Usage: Testing included multiple simultaneous read operations and a mixture of read and write operations. The FSMS processing time remained constant and at the aforementioned averages.

3.3.12 File Deletion and Tape Repacking

Usage: File Deletion from tape was tested and worked properly. Tape repacking was not tested. Recovery of deleted files was also tested. See ACCIDENTAL FILE DELETION - TRASHCAN, below.

³The delayed dismount feature allows a tape volume to remain in the tape drive for a time interval specified by the FileServ Administrator. This feature is designed to reduce tape volume loading time associated with reads and writes to the same volume. The tape can be pre-empted by a file request for a different volume.

3.3.13 Accidental File Deletion - Trashcan

FileServ provides a trashcan delete feature. Full functionality of this function will not be achieved until Version 2.2. Presently, file undelete is **not** yet a supported utility. It is available as an "At Your Own Risk" function. Deleted files remain in the trashcan until the "*fsclean*" function is used to clear the contents of the trashcan and update file and volume status.

Usage: File recovery from the trashcan was attempted and the recovered files could then be successfully accessed. However, certain tape operations would no longer work on volumes with recovered files. Specifically, media removal operations refused to work properly. These problems were reported to the EMASS Technical Assistance Center (ETAC) and the volumes were reformatted. The *fsclean* function was tested and apparently emptied the trashcan.

3.3.14 Media Operations

FileServ provides verbose media monitoring and tracking information in the form of reports and media queries. FileServ will support a partitioned ATL in which only part of the available capacity is available to FileServ. The product allows media to be added and removed from both the silo and from FileServ control. FileServ provides a self-describing media export capability⁴. This allows volumes from one system to be transferred to another FileServ system along with their associated header and directory information.

Usage: Media containing specific files were removed from the silo. Access attempts on these files showed they were unavailable. Replacing the media and attempting retrievals resulted in the appropriate tape to disk copy. FileServ export capabilities were not tested because these functions are only available on DD2 media.

3.3.15 System Logging Functions

The system logging functions were very robust. All aspects of FSMS management are recorded in the log files. A central error log is provided for FileServ. Historical and Trace logs are also provided to track command executions, mount and dismount times, and other FSMS related functions. The system also generates a number of ETAC logs for use by the ETAC. These logs provide extremely fine granularity on all commands generated by the FSMS. The logs include SQL queries to Ingres and status queries to the silo. A "*newlog*" function is supplied by EMASS for manual or automatic close-out of the log files. The log retention time may be set by the FileServ Administrator.

Usage: The system logs were used extensively for hardware and FSMS timing studies during read and write retrievals. The *newlog* function was run from a *crontab* nightly and the retention time was set for 7 days.

Problem: One problem noted with the logs was their lack of time stamp granularity. In developing the FSMS Exerciser Tool for data generation, storage, and retrieval,

⁴ This function is only available on DD2 format media due to the structure of DD2 volume labels and inter-block gaps. This allows the volume to be added to another FileServ System without data reingestion.

it was found that the Convex OS *get_time* call did not return time increments less than one second in a usable format. The usable (hrs:min:sec) provided by *get_time* is the apparent source of the time stamp in the FileServ logs. This lack of granularity was discussed with Convex. They provided several suggestions and code segments to convert the unusable portion of *get_time* to usable data. A conversion method was selected to provide granularity down to milliseconds.

3.3.16 System Reports

FileServ provides a number of useful reports. These reports are accessible via either the GUI or the command line.

Usage: Not all reporting capabilities were regularly used. File, Volume and Class Information reports were frequently accessed during the evaluation. Media reports were less frequently used. A table of available reports and their usage frequency is included in Appendix B.

3.3.17 Degraded Mode Operations

Degraded mode operations allow portions of the associated ATL hardware to be unavailable to FileServ at startup. It also allows hardware to be shutdown during regular operations without fatal impact to the rest of the system.

Usage: The initial testing of degraded mode operations occurred by accident. On two separate occasions an STK drive "missed" its grab of the tape leader. This resulted in a drive with a tape that was neither mounted nor dismounted. In both cases the drive was powered off and the tape was manually removed. FileServ ignored the drive when it was unavailable.

Specific degraded mode testing occurred by powering down specific drives that were not in use. These tests were performed after updating the appropriate parameters in FileServ (to simulate routine maintenance) and without FileServ's knowledge(to simulate a component failure). In each case, FileServ continued operations normally.

3.3.18 Applications Program Interface (API)

Presently, FileServ does not support an API for direct data access. All data queries are either via the command line or the GUI. An API is important because it will allow the development of a control layer for remote data operations. FileServ 3.0 (available 3rd Quarter 94 on the Convex) will support API development.

3.3.19 Identified Problems

3.3.19.1 Problem 1: Error Positioning to End of Tape

There were intermittent problems writing to the end of the tape between 11/2/93 and 11/10/93 during FileServ testing. This resulted in a drive failure report and a log entry of an "error

positioning to end of tape, ioctl error". The error predominantly occurred on volumes containing large numbers of small files.

Technical Support: STK Technical Support was called and a technician came to the site and checked the transport error logs. No problems were discovered with the STK hardware. The ETAC was called and the problem was described in detail. The ETAC was unsure what was causing this problem. See Problem 2.

3.3.19.2 Problem 2: Erroneous Drive in Use Status

FileServ erroneously reported tape drives "in use" on several occasions during the early part of testing (Beginning 11/03/93). The prototype team discovered several instances of tape drive in use according to FileServ, but the STK showed the drive status as available. Physical examination of the tape drive revealed that no tape volume was mounted. Examining the FileServ logs revealed that it still believed the most recently accessed volume was mounted.

Technical Support: Mark Smith at the ETAC was called and this new problem was described. Mr. Smith stated that similar problems were recorded at other FileServ/STK sites. A Technical Assistance Specialist, Mike Sellway, was in the Baltimore Area. Called Mr. Sellway and described the problems we were having. He believed the problems might be due to interface timing problems. Mr. Sellway came to the site and modified some of the STK communications parameters. These modifications temporarily corrected both Problem 1 and Problem 2.

3.3.19.3 Problem 3: Slow Ingres Access and Access Time-outs

It was discovered that FileServ was having problems logging into Ingres. This problem occurred twice, with the first one occurring on 11/03/93. These problems were primarily time-outs while attempting to login. Mark Smith at the ETAC was called to discuss this problem. Mr. Smith said that this was a known problem on the Convex and was related to how the Convex OS uses "nice" values.

Nice values are tunable parameters that allow process priorities to be adjusted. Convex OS uses a combination of "how long a process has been running" and "how much CPU time it has used" to alter processes priorities. Convex performs this operation to balance CPU utilization of all processes in the system. This results in high priority processes being gradually reduced to allow lower priority processes more access to the CPU. Unfortunately, database processes, such as Ingres are CPU intensive, and consequently, require a large portion of the CPU's time. By lowering database process priorities, a login could no longer be achieved prior to a login request time-out.

Mr. Smith stated that it is possible to "*re-nice*" database processes to higher priorities. This worked on the first occurrence of this problem but was ineffective for the second. By the time of the second occurrence, the Ingres processes had been running for 804 hours. Mr. Smith suggested an Ingres shutdown and restart. After the restart, Ingres logins completed normally. Further occurrences of this problem were prevented by periodic restarts of Ingres.

3.3.19.4 Problem 4: Drives Marked Off-line

Concurrent versions of the DADS FSMS Exerciser Program were used for storage and retrievals beginning 11/9/93. Experienced multiple occurrences of "error positioning to end of tape, ioctl error". So many ioctl errors occurred that the tape drive error thresholds within FileServ were exceeded and the devices were marked off-line. Shutdown the concurrent Exercisers and attempted storage operations using a single Exerciser program. The errors persisted. These errors were occurring on volumes with numerous smaller files (1 MB or less in size). The software was apparently unable to position itself to the unused portion of the volume before timing out.

Technical Support: Mark Smith at the ETAC was called and this new problem was described. Mr. Smith stated that similar problems were recorded at other FileServ/STK sites. He further said that this problem appeared to be with the Tape Library Interface (TLI) Daemon in the Convex OS and with the request queue length in the STK ACSSA software.

Called STK Technical Support about a patch to fix the ACSSA queue length (two entries). STK said the most current ACSSA software version was already installed on the system.

Convex Technical Support was called about a patch to fix this problem. Convex had an OS patch release, 10.1.4, currently undergoing beta test. This patch was obtained and installed by Convex Technical Support. No further problems writing or positioning to end of tape were experienced after this installation.

4. Convex Unitree 1.75.15 Evaluation

4.1 Software Configuration

The UniTree software baseline is controlled by a company called Open Vision. Various OEM companies have formal agreements with Open Vision to market and maintain versions of the UniTree product on specific host computers. These agreements allow the OEM vendors to make performance related enhancements to their version of UniTree. The OEM vendors are under no obligation to send performance related enhancements to Open Vision. Other enhancements, such as "bug fixes" are forwarded to Open Vision for inclusion in the UniTree baseline. Open Vision is responsible for functionality enhancements to the UniTree baseline as well as "bug fixes". Open Vision plans a roughly bi-annual baseline release cycle. Once a new baseline is available, OEM vendors require approximately six months for integration and test. In addition, the OEM vendors have the right to determine what portions, if any, of the Open Vision baseline they will incorporate into their version of UniTree.

The second software package selected for review was Convex's version of UniTree. Convex has aggressively marketed their version of UniTree and they have made a number of enhancements to the base UniTree code. The version used in this evaluation, 1.75.14, is a beta release of Convex UniTree +. Convex has made a number of enhancements to increase the performance of their UniTree product. A Software Product Specialist was sent to site for the UniTree installation. Convex also provided training materials and 3 days of Software and Administrator Training. Several configuration and hardware problems were encountered during the evaluation. These problems were amplified by the Winter Holiday season. Several functions were not tested due to the time lost for these problems. Appendix E details the major problems experienced during the evaluation.

The UniTree product is different from FileServ in that data is not resident in UFS. Data resides either in the UniTree Disk Cache (part of the UniTree File System) or in secondary storage under UniTree control. Access to the data was somewhat more difficult than in FileServ. The user/data interface to UniTree is either via a File Transfer Protocol (FTP) or a Network File System (NFS). User to UniTree interactions are limited to the functions available in FTP or NFS.

UniTree is a very administrator intensive system. Parameter modifications are not straight forward and usually require modifications to several files. Some daemons must also be restarted for certain changes.

The following UniTree features were evaluated:

- General
- Startup/Shutdown
- Placing Files Under UniTree Control
- File Migration And Staging

- Disk Cache Management
- Ease Of File Ingest
- Concurrent File Write
- Ease Of File Retrieval - Sequential, Random, Partial
- Concurrent File Read
- File Deletion And Tape Repacking
- Accidental File Deletion - Trashcan
- Media Operations
- System Logging Functions
- System Reports
- Degraded Mode Operations
- Applications Program Interface (API)

4.2 Convex UniTree: Enhancements

4.2.1 Regular File Disk Cache

UniTree can be configured to read and write to a disk in a regular file format in the Convex OS. This type of disk cache allows a large amount of write behind to improve FTP and NFS write performance, buffer cache read-ahead to improve NFS read performance, and more efficient use of memory on large memory machines with disk reads minimized in FTP transfers involving the caching of data from tape to disk.

4.2.2 Rcp Support

Users are now able to copy data into the UniTree+ disk cache via the BSD Unix Remote Copy function (RCP).

4.2.3 Volume Mount-Ahead

The UniTree *tapesrvr* now mounts the next migration volume⁵ when it determines that the current migration volume will not hold all the data being migrated this round.

4.2.4 3480 Fast Seek

File to tape location addresses are stored in a separate database maintained by the *tapemovr*. Indexes from the database are passed to the *tapesrvr* header database for storage with each fragment of the given file. Fast seek utilizes the STK "*locate block*" channel command and performs tape searches on two levels. First, a high speed search to the general vicinity of the

⁵ Data entering UniTree is always associated with a file family. A migration volume is a tape associated with the file family of the migrating files.

desired record, then a normal reading speed search to the desired record. This feature is advertised for the 3480. It is assumed that it will work on any STK form factor capable of working with the *locate block* channel command.

4.2.5 Logical Volume Dismounts

Tape mount and dismount requests are evaluated against any outstanding tape mount requests. New requests for data on the mounted tape will be processed immediately. If no new requests are queued, the tape will remain mounted for a configurable period of time or until it is preempted by an active data request.

4.2.6 Parallel Copy Writes

UniTree writes the primary instance of each file and any associated copies at the same time. There is normally a one file difference in writes. (i.e. the primary transfer will be one file ahead of any duplicate copies being written.)

4.2.7 Parallel tapemovrs

Multiple copies of the *tapemovr* are spawned by *tapemstr* at startup. This number is based on the available drives. Each *tapemovr* can process requests independent of the other *tapemovrs*. This allows concurrent I/O. Presently, multiple reads and parallel copies are possible. Concurrent writes were not available for this release.

4.2.8 Namesrvr Performance

The *name server* has been tuned and optimized so it is less CPU intensive. This results in increased NFS performance.

4.2.9 Repacker Disk Cache and Tape Drive Throttling

Tunable parameters are now available to limit the proliferation of file stage requests from the tape repacker to the disk server and also limit the number of available devices used by the repacker.

4.2.10 FTP Mstage

UniTree+ FTP functionality has been expanded to allow multiple files to be staged to disk using the *quote mstage* command.

4.2.11 UniTree Path Sockets Parameter

The SOCKETS parameter in "*UniTree.path*" can be used to configure the directory in which UniTree creates the Unix domain sockets used for interprocess communication.

4.2.12 Additional Parameters in UniTree.Conf

- diskserver.nfs_async_writes* - - Enable or disable asynchronous UniTree/NFS writes.
- diskserver.nfs_use_sticky_bit* - Bit used to denote archival files in an NFS directory.
- pvrserver.dsmnt_retry_interval* - This Parameter control how often to retry failed dismounts (in seconds).

4.3 Convex Known Problems

4.3.1 More than 6 Ultranet Connections

This is a problem with Ultranet that Convex is currently working to solve. If more than six (6) Ultranet connections are established, UniTree and the ConvexOS will hang. Both ConvexOS and UniTree must be rebooted.

4.3.2 Automatic Tape Repacker

Approximately 90% of the time, the automatic repacker does not know if it has tapes to repack. This problem is magnified if multiple copies of files exist and/or if file families are used.

4.3.3 Disksrvr Startup Panic Message

Frequently, the *disksrvr* will fail to start properly when the system is booted. After the initial startup "panic", the persistent daemon retries and is usually successful.

4.3.4 Mass Staging (*mstage*) Problem

The mass or multiple staging command in FTP may hang if an *mstage* operation retrieves files from multiple volumes.

4.3.5 Full Disk Cache Problem

FTP may hang if a staging command is issued on a file that exceeds the size of the disk cache or the disk cache fills up prior to completion of the staging operation.

4.4 Data Distribution

Table 4-1. UniTree Evaluation Data Distribution

Family Number	# of Files	File Size(s)	# of Tapes Used
14	1	1 GB	6
15	1	500 MB	3
16	1	300 MB	2
17	5	200 MB	5
18	7	100 MB	4
19	10	10 MB	1
20	100	1 MB	1
21	200	100 KB	1
22	1000	80 KB	1
23	1000	64 KB	1
24	2000	16 KB	1
25	2000	1 KB	1
TOTAL	6325 files		27 volumes

The above data file distribution was used for the UniTree evaluation. UniTree file families are indicated by a numeric reference as opposed to the alphanumeric reference possible in FileServ. The number of files in each family and the associated file sizes have been included for reference.

Problems were experienced ingesting data into UniTree. Part of this was due to network problems and a Service Processor Unit (SPU) disk corruption which resulted in a significant loss of time. Other problems related to the degree of file control possible under UniTree, and the organization of the storage and retrieval tests. In addition, the fact that UniTree keeps files in a separate File System from UFS, amplifies these control problems. The sections below will provide more information on specific problems.

4.5 Operational Evaluation of Features

4.5.1 General

For the purpose of this evaluation, NFS access methods were not used to any great extent. Both Convex engineers and other UniTree sites indicated that this method was much slower than FTP access. A number of the enhancements to UniTree + were designed to increase NFS performance in the Convex UniTree environment. Plans were made to test the NFS access capabilities of UniTree+ but resolution of a system problem exhausted the time allocated for this testing.

4.5.2 Startup/Shutdown

UniTree starts when ConvexOS comes up after a shutdown. UniTree Daemons and Servers are created for the various UniTree processes. Convex is using persistent daemons. A persistent daemon will restart if it abnormally terminates. An abnormal termination usually results in a core dump (*ucore*) for the specific process. There is a danger that the UniTree file space can become filled with *ucores* during software problems due to the persistence on the daemons. NFS mounting of the UniTree Disk Cache will also occur at startup. The NFS mount will lock the console for several minutes while everything initializes.

Usage: UniTree can also be stopped and started at times other than ConvexOS initialization. A shutdown of a quiescent UniTree system takes 20-30 seconds. A UniTree startup will take 2 - 3 minutes. Some problems experienced with UniTree require a shutdown while others can be remedied by stopping the appropriate server.

4.5.3 Placing Files under UniTree Control

It is relatively easy to place files under UniTree control, however, transparent access to secondary storage can only be achieved using NFS. An FTP interface requires a file copy operation from UFS to the UniTree Disk Cache. Thus, in the worst case, a file can be resident in three places simultaneously. (i.e. UFS, the UniTree Disk Cache, and the ATL.)

Usage: An FTP connection is made to the local host. Once the connection has been established, the user may set the family he wishes transferred data to enter. Then a "put" operation is performed and the file is now resident on the UniTree Disk Cache. Migration and staging operations are performed automatically by UniTree.

4.5.4 File Migration and Staging

Staging operations occur on demand. File Migration occurs based on four configurable parameters. These parameters determine which files are migratable and when migration will occur. A file must be resident in the UniTree Disk Cache for a specified number of minutes before it is eligible for migration. The migration server checks at intervals (established by the UniTree Administrator) to see if the file count threshold has been reached and the migration wait time has been exceeded. (These parameters are also under administrator control). Migration parameters can be changed while UniTree is running, but they will not go into effect until the next reboot or until the migration server is stopped and restarted. The UniTree 1.7 baseline has provided a manual migration utility, "*forcemig*", to force file migrations. Forcemig is designed to work on groups of files. Parameters include all files or specific file groups based on age⁶.

After a file is migrated it is eligible for purging. The purging algorithm can be weighted either for file size (larger first) or by time (based on a least recently used algorithm.) Purging is also controlled by high and low watermark parameters. Convex has supplied a manual purge utility, "*forcepurge*", that functions on a single file. A force purge operation can only be performed by a superuser.

⁶ Age refers to the length of time, in minutes, that a file has been in the disk cache.

Usage: During the initial stage of testing, force migration and force purge operations were performed to facilitate ingest and retrieval testing. These commands were somewhat cumbersome because specific files could not be directly controlled. Later the migration and purging parameters were altered to greatly reduce a file's residency time on disk.

4.5.5 Disk Cache Management

Usage: The UniTree Disk Caches were built as regular file disk caches (See ENHANCEMENTS above). There was no easy way to manage the cache. The ConvexOS "*df*" command showed each cache was 100% full because it was built as a regular file. UniTree does not actively manage its disk cache. There are purge criteria established as high and low watermarks but these watermarks are not continuously monitored. It is possible for the UniTree Disk Cache to overflow. The largest files used in this testing were 1 GB. The three disk cache partitions were each over 1 GB in size. It is unclear what would have happened if a file greater than the size of any available disk partitions were ingested.

4.5.6 Ease of File Ingest

Files enter UniTree either via an NFS mount of the disk cache or via FTP access to the disk cache. UniTree uses the standard FTP commands that come with ConvexOS as well as some UniTree specific enhancements. See Appendix D for a list of commonly used ConvexOS and UniTree + FTP commands.

Usage: The NFS mounted disk cache was tested during UniTree training. This feature seemed to work but specific performance, and multi-user tests were not performed at this time. The FTP access method was used during the product evaluation functions in UniTree + were helpful for ingesting and retrieving groups of files.

Problem: The NFS mount of the UniTree Disk Cache at system startup will lock the system console for several minutes. Convex is aware of this problem.

4.5.7 Concurrent File Write

This function was not available for normal file ingest in this version of Convex UniTree. A near simultaneous capability to make backup copies does exist.

Usage: The concurrent capabilities inherent with backups was not tested due to the need for specific quiescent system storage rates. Plans were made to test this feature later, but system problems precluded this testing.

Problem: The initial data rates reported for file writes & reads were very disappointing. Convex was consulted about this problem. An error was discovered in the algorithm that calculates the data rates. Debug checkpoints were turned on in the software to provide an accurate picture of UniTree's performance.

4.5.8 Ease of File Retrieval - Sequential, Random, Partial

File retrievals were accomplished via the FTP interface.

Usage: Sequential files were retrieved using the "*mget*" and the "*mstage*" commands. Random retrievals were performed via the "*get*" command. UniTree only supports whole file retrievals.

Problem: The *mstage* command worked well with delayed dismount. Entire blocks of files could be read from tape without interruption from mount and dismount operations. The *mget* command seemed to have a built in dismount command. When *mget* was used for a sequential retrieval, the selected volume was mounted and dismounted between each file retrieval.

4.5.9 Concurrent File Read

Usage: This version of UniTree supported concurrent read requests. This function was tested and worked well. Files from different families were retrieved to simulate multiple user requests. No measurable performance difference were discovered.

4.5.10 File Deletion - Trashcan

There was insufficient time to examine File Deletion and Repacking issues with this version of UniTree.

4.5.11 Accidental File Deletion - Trashcan

There was insufficient time to examine the functionality of the *trashcan* delete feature.

4.5.12 Media Operations

UniTree will allow the administrator to change a tape's location indicator in the tape map. This allows a tape to be removed from the ATL and be placed in a manual "*vault*". Vaulted tapes must be manually loaded and removed from an ATL. Currently, Convex UniTree does not track request to files in the manual archive. If a mount request is made for a vaulted tape the request is sent to an ATL and UniTree assumes the appropriate tape will be made available.

Usage: ECS Requirements do not allow for tape *vaulting*. This fact, coupled with a shortage of time resulted no evaluation of the *vaulting* capabilities of UniTree.

Problem: This version of Convex UniTree does not effectively manage the available tape drive resources in the ATL. This was specifically noted with the STK Wolfcreek attached to the Convex. Queries were always routed to the same tape drive even though four drives were available to UniTree. This places excessive wear on a single device. Other devices were used as needed but this seems to be a deficiency from a Preventive Maintenance viewpoint. This was discussed with Convex UniTree support during UniTree Training. Convex has plans to address this problem in a future release.

4.5.13 System Logging Functions

The UniTree log files are difficult to use and generally poor. Each server and some utilities have individual log files. Many of the logs have different formats, and the logs refer to file events using the UniTree Capability ID⁷ instead of the "human file name"⁸ (hname). Another problem is that these logs continue to grow indefinitely. Log management is a fundamental administrator function. Convex has begun to address this problem by including a date/time stamp with each entry. Thus, events can be tracked between log files. Convex plans to consolidate the UniTree logs into a single central logging facility in a future release.

4.5.14 System Reports

System reporting capabilities are limited in UniTree and, in general, these were not tested. A nameserver report resolving Capability ID with hnames was used on several occasions to track specific storage operations.

4.5.15 Degraded Mode Operations

We manually declared tape drives unavailable to simulate degraded mode operations. This process required modifications to several parameter files and a restart of a daemon. Degraded mode operations without UniTree's knowledge were not tested.

4.5.16 Applications Program Interface (API)

Not available in this release of UniTree.

⁷ A capability ID is a UniTree generated name used by UniTree for storage, retrieval, and other file operations.

⁸The human file name is the name given to the file by a user.

This page intentionally left blank.

5. Conclusions

These conclusions are intended to summarize the observations and experiences of the prototype team during the operational evaluation of FileServ 2.1.6 and UniTree 1.75.14. The evaluation clearly indicates that FileServ is superior to UniTree in several areas. This is particularly true within the boundaries of the Evaluation and Test Environment. Testing in a Production Oriented Environment is needed to validate these differences in an operational configuration.

The primary disadvantage of UniTree is that it manages data in a separate file system from UFS. This means that Unix scheduling and error functions utilized by FileServ are unavailable to UniTree. A specific example is disk cache management. When a disk partition is created in the *fstab* file of Convex OS, high and low watermarks for this partition are established. Unix monitors these watermarks and provides a general system message when watermarks are reached. These messages are used by FileServ to manage the disk cache resources of partitions under FileServ control⁹. UniTree must monitor its own disk partitions using watermarks established in its parameter lists. Unlike FileServ which has virtually continuous monitoring because it utilizes UFS, UniTree schedules these monitoring sessions using timers. (e.g. Test the disk cache every 10 minutes.) Potentially, the UniTree disk cache can overflow and data can be lost. This function was not explicitly tested due to the data entry methods used for this evaluation¹⁰, but the potential problem illustrated by this example should be clear. Unix resident FSMS products can off load some scheduling, timing and monitoring functions to Unix, which is running all the time. UniTree must, for the most part, explicitly manage storage resources and the associated data, in addition to scheduling and monitoring system resources.

Another advantage of the FileServ product is the relative ease with which system parameters can be modified. Most FileServ parameters can be accessed via the Motif GUI, the command line, or by editing a single file. UniTree parameters normally require edit operations on multiple files and the software does not notify the user or the administrator of parameter mismatches. (A problem we saw during testing.) In addition, UniTree required more of an administrator's time. FileServ was relatively self-sufficient, a desirable feature from an ECS operations & management standpoint.

Error logging in FileServ is centralized and a single log file contains all errors for each logging period. Separate logs track ATL operations and user requests. UniTree's logging functions are decentralized with each process or daemon maintaining its own log file. Formats of these logs vary and the files must be manually managed. (i.e. the process and daemon logs will continue to grow indefinitely, other logs will wrap around after "x" number of lines have been written.) The lack of centralized logging hampers performance monitoring and fault isolation.

⁹ A disk partition or portion of a disk partition is placed under FileServ control by establishing a DataClass to Unix directory relationship.

¹⁰ Data was ingested and then immediately migrated to tape. The disk copy of the file was then truncated. Data did not remain resident of the cache for long periods of time. This was an attempted to simulate a "high ingest, process later" environment.

FileServ provides the user with a number of reports and queries that provide system resource and status information as well as specific file information (e.g. creation date, size, dataclass, etc.). File tracking information (e.g. what volume a file resides on, volume location, etc.) is also easily accessible within FileServ. This information is provided based on the name assigned by the user (hname). UniTree provides some system and file information but UniTree maps the hname to a UniTree Capability ID. The Capability ID is used for all file management operations within UniTree and has known significance with regard to the hname¹¹. This increases file identification and tracking problems by an order of magnitude due to the additional mapping of the Capability ID to hname. Something ECS wishes to minimize.

A major limitation noted with Non-NFS access methods to UniTree is that a file will exist in two places. The file resides in a UFS directory and it must be moved, via RCP, FTP, etc., into the UniTree disk cache. This movement requires a network connection and transfer, even if the file and the UniTree System are on the same host. The data must then be moved in some manner to the ATL. FileServ can manage UFS files directly, thus eliminating the need for several file transfers. In addition, FileServ can manage each file individually. (i.e. a user can manually move, copy or truncate a single file. UniTree operates on groups of files almost exclusively.) Such limitations may degrade the performance of ECS.

A final advantage of FileServ is its ability to function in a degraded mode. FileServ will allow devices to be shutdown either for periodic maintenance or due to failure, and it will boot even if some ATLs or other resources are unavailable. Some versions of UniTree will not boot if all devices listed in its configuration table are not available. Periodic Maintenance is possible with UniTree but an elaborate process is required to insure the device does not receive and queue data requests when it is shutdown, and that it will be available to UniTree when maintenance is completed. The impact of device failures varies between the different OEM versions of UniTree. A limitation of UniTree from a device standpoint, is that devices are assigned to UniTree when the software is initialized. Making devices temporarily unavailable to UniTree (e.g. allowing users or system operations to use one or more of these devices for backups or distributions), is non-trivial. FileServ is designed to check device availability via Unix. If a device is available when a request is received, FileServ will use it. If the device is in use by some one else. (i.e. system operations or the system administrator), request are either queued or another device is selected. This flexibility is very attractive from an ECS perspective.

As previously mentioned, these advantages were discovered using a nearly quiescent Test Environment. A more Production Oriented Test is required to validate these observations.

¹¹ The Capability ID is assigned based on a UniTree algorithm. The assigned ID has no relation to the hname. (e.g. a Capability ID of JOEDATA123X is not necessarily the hnamed file JOE.DATA.)

6. Ancillary Data

Table 6-1. FileServ Command Summary (1 of 3)

FileServ [-c -y manual tower silo -r -y manual tower silo -t -y]	Terminate or activate FileServ software
fsaddclass <class> [-s -h -n -i -o -x -d -t] [-py/n] [-rc/s] [-m]	Create a new data class.
fsaddrelation <directory> [-c]	Associate a directory to a data class.
fsaudit <filesystem> [-f / -i] <newfilesystem> [-r] [-f / -f] [-o -d]	Audit a file.
fschfiat <filename> [-c -e -c -i -c -[-c]	Modify the class attributes of a file.
fschmedstate <medialD> [-s]	Change the state of a medium
fschstate <componentalias> [-s -u]	Change or report the state of an EMASS hardware component.
fsclassinfo <class> [-l]	Report data classes, their processing parameters, and directory paths within each data class
fsclassrm oldclass newclass	Rename the identifier of an existing data class or merge one class into another.
fsclean -c -m <medialD> [-t -c -t]	Remove the “trashcan” information
fsrminfo -x -b <medialD>	Remove file and media information from FileServ Software
fsrmrelation <directory>	Remove a directory-to-data class relationship
fsrolldown <directory>	Move the directory-to-data association down a directory level.
fsrollup <directory>	Move the directory-to-data class association up a directory level.
fsstore <filename> [-i -t -c] [-f i / p]	Expedite the storage of a file, that currently resides on EMASS disk, to EMASS media.
fstddump -AOES -t <starttime endtime> [-i] <logID>	Extract tape statistics information from the EDAC logs.
fstsrpt [-c / -AOES -T]	Extract information from the EDAC logs.

Table 6-1. FileServ Command Summary (2 of 3)

fsvolpath <volumeID>	Find the path name for transcription files.
fswin	Display the System Administration Graphical User Interface console.
fsconfig [-a -h -m -d] <component ID> [-i -t -r -v -s]	Configure or report EMASS hardware components.
fsdirclass <directory>	Report the data class associated with a directory.
fsdump <key argument filesystem>	Perform an incremental file system dump.
fsexthlog <logfile> [-t] <endtime>	Extract the contents of the system log file.
fsfileinfo <filename>	Find a file in the EMASS system and list non-media specific information for a file in the EMASS system.
fsgetclasses <filesystem>	Report all classes with association points in a file system.
fshistrpt [-h] <componentID> [-t -m -t]	Report history of hardware component state changes or media status changes.
fsmanual <controlleralias>	Activate the Manual Storage Subsystem controller.
fsmark <mediaID> [-c -k -n -u]	Mark media for removal from the storage subsystems.
fsmedconfig [-t] <mediatype> [-a -m -d -f -l]	Change or report the configuration of the media types available for use by the EMASS subsystems.
fsmedcopy <mediaID> [-e -r -u] [-t] <mediatype>	Copy or defragment data by medium.
fsmedin [-d -q -i -t -m -h -f]	Enter media to an EMASS storage subsystem.
fsmedinfo <mediaID> [-l]	Generate a report on media based on their current status.
fsmedlist [-v -c -i -l kxembpausdhtzo -g]	List media in a data class and/or storage area.
fsmedloc <mediaID> [l]	Change the external location for media.
fsmedout [-q -b -i -c] <mediaID>	Remove media from the EMASS storage subsystems.
fsmodclass <class> [-s -h -n -l -o -x -d -t] [-py / n] [-r c / s] [m]	Modify the processing parameters of a data class.

Table 6-1. FileServ Command Summary (3 of 3)

fsmovept - abxmoi <mediaID> [-t]	Generate a report of the media that have been removed from or introduced into EMASS storage subsystems.
fspolicy -s <class> [-t -b]	Apply the specified policy.
fsqueue [-c]	Cancel or view subsystem resource requests.
fsreprintlabel <mediaID> [g]	Reprint a label for a list of media.
fsrestore [t x v o r R] [Gibfshvy Gibfhmsvy Gibfsvy] <filename>	Rebuild a migration file system after a disk crash.
fsretrieve <filename> [-n -b -f -n -c -f]	Retrieve a file or recover its secondary copy from an EMASS medium and place it on disk.
fsrmclass <class>	Delete a data class.
fsrmdiskcopy <filename>	Remove the disk copy of a file after the file has been stored to tape.

Table 6-2. FileServ Queries and Reports

QUERIES AND REPORTS	USAGE - (very frequent - frequent - seldom)
data class information query	very frequent
relation information query	frequent
file information query	very frequent
media information report	very frequent
media list report	very frequent
media external location report	seldom
media defragmentation report	seldom
media movement report	seldom
resource queue report	seldom
history report	frequent
component statistics report	frequent
tape statistics report	seldom
tape statistics dump	seldom

6.1 FileServ Error Log

Date:11-01-93

Entry: During file storage we experienced a drive failure. The error was an error positioning to end of tape. The drive was not available for use and the media was marked suspect¹². This error was not detected until 11-03-93 because we were storing one file at a time during this period so other drives would be used and the drive "in use" was skipped when FileServ was sequencing through the drives. We also received sporadic problems connecting to the database.

Date:11-02-93

Time: During an overnight run of the FSMS Exerciser.

Entry: During file storage we experienced a drive failure. The error was an error positioning to end of tape. The drive was not available for use and the media was marked suspect. We also have been having trouble connecting to the database.

Contact: Called Convex Technical Support to find out about killing and restarting the TLI daemon to solve the drive problem. Also called STK to see if they could determine anything about the drive problems. They sent someone out who looked at the logs, determined that there had been no errors since 23 September. Was contacted by Mike Sellway of EMASS who was at another site. He came out and modified STK timing parameters used as time-out values and wait times. It was not determined what effect these changes had on the system because we still encountered problems.

Date:11-03-93

Time:3:30 time of last lock-up

Entry: During file storage we experienced erroneous *drive in use* indications. Also noted some Ingres login failures.

Contact: Called Mark Smith of EMASS. He suggested an interim solution to us. He told us to run a utility called `update_media` and that would reset the database so that FileServ would release the drive and reset itself so the media could be used again. He also suggested we *renice* the database to solve the Ingres problems. He said that the way Convex OS deals with processes and priorities causes problems for a process that runs for a long time.

Date:11-04-93

Entry: During file storage we experienced erroneous *drive in use* indications. Also received several errors positioning to end of tape, `ioctl` error. To this date we have received this error approximately a dozen times. We are using the suggested utility `update_media` to reset FileServ.

¹² A suspect tape will allow normal read operations to occur but further write attempts on the suspect volume are prohibited by FileServ.

Date:11-08-93 Time:4:00 PM

Entry: We logged database access errors when storing files. We tried to renice the database processes but it did not work.

Contact: Called Mark Smith of EMASS. He asked us to check the Ingres process system time. We found it to be 866 hours. Mr. Smith said that a renice at that point would not help because of the high system time. He suggested that we shutdown FileServ and Ingres and restart them. By restarting Ingres we noticed a significant increase in response time.

Date:11-09-93

Entry: Again experienced the tape positioning to end of tape ioctl error. This time along with marking the media suspect it began to mark the drives off-line one by one as the error counts increased. These errors seemed to increase dramatically when we started using 4 concurrent processes to store files

Date:11-10-93 Time:11:10 AM

Entry: During a file storage operation we experienced a drive failure report on drive /dev/rtd1n, media EVL178 during a write operation, writing file label EOF3, error : I/O error.

Contact:Talked to Mark Smith of EMASS. He told us that they have experienced problems at 2 other sites with the tape positioning to end of tape error. They seem to think that it is a time-out problem and that Convex has a patch to their OS, specifically the TLI daemon that will resolve these problems. The patch is patch 10.1.154. Mr. Smith also said that version 10.1.4 of Convex OS will also resolve this problem.

Date: 11-11-93 Time: 10:36 AM

Entry: During data retrieval we experienced a read error on volume EVL009 reading file /fsms/classa/ FS000010.DAT. The tape was then marked suspect. A second retrieval attempt was made on the file. No error was generated and the file data was verified to be intact.

Table 6-3. FTP & UniTree FTP Commands (1 of 2)

aappend local-file-remote-file	Appends a local file on the remote machine.
ascii	Sets the file transfer type to ASCII.
binary	Sets the file transfer type to support binary image transfer.
bye	Terminates the FTP session with the remote server and exits FTP (SAME AS QUIT).
cd remote-directory	Changes the current directory on the remote machine to the directory specified.
cdup	Changes the current remote machine directory to its parent directory.
close	Terminates the FTP session with the remote server and returns to the command interpreter (the FTP prompt).
delete remote-file	Deletes the specified remote file from a remote directory.
dir [remote-directory]	Lists the entries and entry attributes of a directory for the remote machine.
get remote-file [local-file]	Copies a remote file from a remote directory to the local machine.
hash	Toggles the printing of the pound-sign (#) on the screen for each data block transferred.
help [command-name]	The help command by itself gives a list of valid FTP commands.
lcd local-directory	Changes the current directory on the local machine.
ls [remote-directory]	Lists file names in a remote directory.
mdelete remote-file1 remote-file2	Deletes multiple files. This command deletes the specified files.
mkdir remote	Makes a directory on the remote machine.
mput local-file1 local-file2 local-file3 ...	Copies multiple local files from one or more local directories to the current remote directory.
open host [port]	Establishes a connection to the specified host (remote) FTP server.
prompt	Toggles interactive prompting.
put local-file [remote-file]	Copies a local file from a local directory to the remote machine.
pwd	Prints the name of the current remote directory.
quit	Terminates the FTP session with the remote server and exits FTP.
quote arg1 argue ...	The arguments or commands specified are sent verbatim to the remote FTP server.
recv remote-file [local-file]	A synonym for get.
rename from-name to-name	Renames the file from-name on the remote machine to the file to-name.

Table 6-3. FTP & UniTree FTP Commands (2 of 2)

reset	Clears the reply queue.
rmdir remote-directory	Deletes a directory on the remote machine.
send	A synonym for put.
site quote site UniTree+_specific_FTP command	This command is used by the server to provide services specific to its system that are essential to file transfer but not sufficiently universal to be included as commands in the protocol.
status	Shows the current status of FTP.
user [login-name] [password] [account]	Identifies you to the remote FTP server as the same or a different user.

Table 6-4. Convex UniTree Unique Commands

GTRSH	
STRSH minutes	Displays the trash can timeout interval for the user in minutes
NMDUP [n]	Changes, permanently, the trashcan timeout interval for a user.
SETFAM [family_name]	Sets, for the session, the number of multiple instances to be stored on tape.
chgrp group-name filename	Sets, for the session or until SETFAM is re-executed, the family name to which files will belong.
chomd	Changes the group associated with a remote file.
chown	Changes the permissions of a remote file.
In file1 file2	Changes the owner of a remote file.
unmask permission-mask	Allows you to create a symbolic link.
stage waittime path	Sets, for the session, the file creation mask for default permissions on new files.
	Caches a file from tape to the disk cache. If you request a file that is stored on tape, it may take time for UniTree+ to retrieve the file. waittime is the amount of seconds you want to wait for the prompt to return. FTP get will attempt to retrieve a file to the client disk from tape.

6.2 UniTree Error Log

Tracking errors with UniTree+ was a difficult task because a central logging facility was not available. In addition, the evaluated version of UniTree+ was still in Beta Test. Some errors that came up on the console were due to problems that Convex plans to resolve prior to release of UniTree+. Other errors proved to be debug messages that would not appear in a production release.

For the most part, error messages were ignored. The only real problems experienced during the evaluation are documented below.

6.2.1 Problem 1 - SPU Disk Corruption

The Convex 3220 was down for approximately four (4) days beginning 17 December 1993. During the mid-December timeframe, DCE software was being loaded on the Convex host in preparation for support of Evaluation Package Three (EP3). Most of December 17th was spent troubleshooting the second Ethernet interface on the Convex Host for EP3. Prototype personnel noticed that the Convex would not allow users FTP access to UniTree. Examination of the system process table showed that multiple "asiodaemons¹³" (over twenty-five) were running. In addition, modifications were needed in the /etc/fstab file to allow NFS testing to begin.

Local Convex technical support was called and they suggested a system reboot to resolve the FTP problem. The reboot was also required to implement the /etc/fstab changes for NFS testing. A system shutdown was initiated with a reboot option. When the system rebooted, multiple UniTree error messages began to appear on the console (due to persistent daemon execution). The UniTree file system was soon full. The reboot process was halted and the UniTree File System was cleaned up. Examination of the system process table showed multiple asiodaemons (ten or so) were still resident. All UniTree servers and daemons were manually killed from the system process table. A second shutdown with reboot was performed. The same type of UniTree problems occurred upon reboot.

After the system process table was cleared a shutdown to a halt was performed followed by a manual reboot. The same errors were experienced. The system was halted a second time and the decision was made to power off the CPU to clear memory. Upon power up the Convex would not even boot to the SPU disk level. Convex Technical Support determined that the SPU disk was corrupted. SPU disk updates are kept in memory and written to disk periodically. The CPU was halted prior to a periodic SPU disk update. Convex Technical support indicated that a *pwrdown* command should be issued prior to interrupting system power. This command forces a *sync* operation on the SPU disk to insure it is ready for shutdown.

The disk was reformatted and reloaded with new software. The Convex host booted properly after the reload.

¹³Asynchronous Input/Output Daemon.

6.2.2 Problem 2 - Drive Parameter Mismatch

Once UniTree was back up on December 21st, testing was resumed. A problem was immediately discovered with the ATL. UniTree appeared unable to command the silo. ConvexOS silo commands and STK ACSSA Software commands appeared to work fine. Convex Technical Support was called and began examining the UniTree configuration files. Parameter mismatches were discovered in several files. These mismatches were corrected, the system was restarted, and testing resumed.

6.2.3 Problem 3 - Inaccurate Transfer Rates

Earlier in the evaluation period, the prototype team complained to Convex about the extremely low transfer rates coming from the migration server statistics. Initially, the thought was a tuning parameter was set improperly. Convex came to us on the 22nd of December with news that the algorithm being used to create migration statistics in *migsrvr.t* was not configured properly.

This invalidated the transfer rates collected to date. It was decided that data ingestion would begin again and these tests would include the 128k extended blocking factor available in UniTree+.¹⁴ Convex Technical Support also turned on some debug statistics that would give us accurate transfer rates for read and write operations. Some table and configuration problems were experienced while allocating families to the 128k extended blocking factor. In the end, only one family was formatted with this blocking factor.

¹⁴This blocking factor was primarily intended for use with the UniTree + interface to Metrum SVHS devices.

This page intentionally left blank.

Abbreviations And Acronyms

ACSLs	Automated Cartridge System Library Server
ACSSA	Automated Cartridge System System Administrator
API	Applications Program Interface
ATLs	Automated Tape Libraries
BSD	Berkley Software Distribution
CPU	Central Processing Unit
DCE	Distributed Computing Environment
EMASS	E-Systems Modular Automated Storage Systems
ETAC	EMASS Technical Assistance Center
FSMS	File Storage Management Systems
FTP	File Transfer Protocol
GB	Giga Byte
GUI	Graphical User Interface
Inodes	Index Nodes
KB	Kilo Byte
MB	Mega Byte
NFS	Network File System
OS	Opening System
RCP	Remote Copy
SPU	Service Processor Unit
SQL	Standard Query Language
STK	Storage Technology Corporation
TLI	Tape Library Interface
UFS	Unix File System